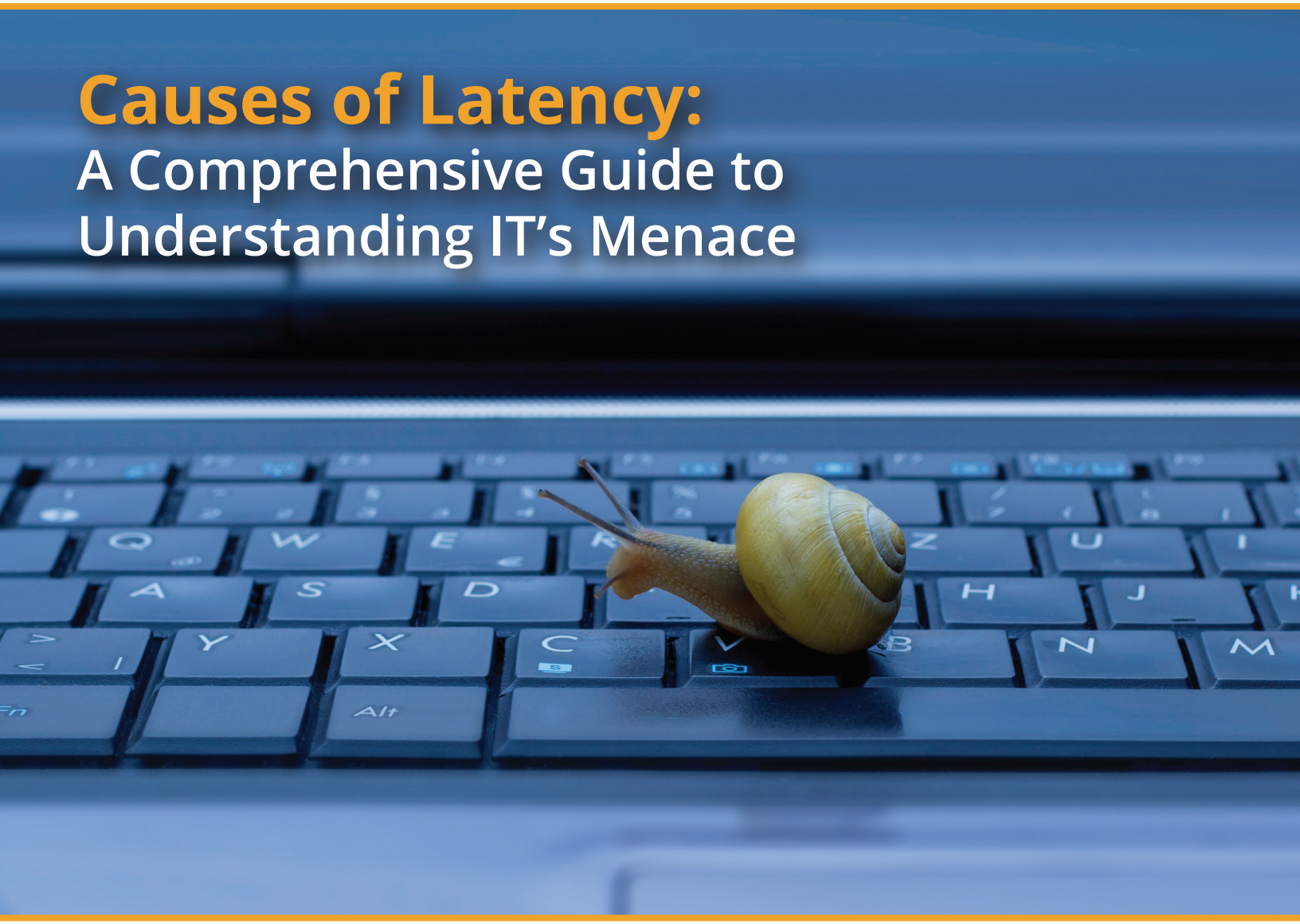
Causes of Latency:
A Comprehensive Guide to Understanding IT's Menace



Mirazon

Content

Intro 2

Interconnects 2

Network: 6

- Basics and history 6
- Uplinks 8
- Routing 8
- Internet 10

Servers 11

- Hardware Concerns 11
- Operating System Concerns 12

Storage 13

- General 13
- Disks 14
- SSDs 15
- RAID 15
- Other features and concerns 16

User Experience 18

- Local on premise user: 18
- Long Distance (point to point/MPLS/VPN) 20
- VDI/Remote Desktop/Terminal Servers 21

The Cloud 22

Publicly hosted servers and applications: 23

Conclusion 22

"Mirazon has improved our IT by not only being a fresh set of eyes to look at a problem but also the engineers that work there are just a knowledge vault and they know their stuff. Every single guy at Mirazon that I've talked to is an A-team player and because of that, they can solve issues quickly and completely in a manner that those issues don't have to get revisited.**"**
– Dave Mast, NewPointe Community Church

Intro

Latency sucks. In IT, pretty much every "speed" problem that we run into is actually a latency problem. The purpose of this article is to try to refine critical thinking regarding what "this isn't going fast enough" really means. For starters, what is latency? One of the best definitions from dictionary.com (that doesn't involve a tautology) is, "The period of apparent inactivity between the time the stimulus is presented and the moment a response occurs," or, translated: "the length of time between when you poke the bear and when it catches you."

Disclaimer: I'm not an electrical engineer, physicist, mathematician, or any of the other over-educated people who truly understand the complex and variable-rich math involved with some of the calculations referenced in this paper. Some of these numbers took hours of research and reading far more about physics than I cared to, simply to get something as straightforward as "the industry states that a good rule of thumb is .66." All of the numbers in this paper which I acquired are preceded with the following cryptic symbol: "AROUND." So, don't throw things if you don't agree with a number.

We'll approach latency at a few different places:

Interconnect
Network
Server
Storage
THE CLOUD!
Users (their experience, not the squishy part in the chair that whines a lot)

Interconnects

Interconnects get a bad rap. They are blamed (often erroneously) for a lot of latency they do not cause directly. Not to say that they can't add latency, but it's often the ends of the connection that are the true culprits, not the bit in between. To see why, let's start with some physics and math (bear with me, we'll round to the nearest thousand).

For starters, we have the speed limit. The fastest most people agree anything can travel is the speed of light: AROUND 186,000 miles per second symbolized by c (as in E=mc2). Yes, I meant second; not per minute, not per hour, but per second. Light traveling in a vacuum could circle the earth 23 times in one second. The problem is that anything done with light results in slowing it down. Stick it in our atmosphere (in the form of RF) and it slows down to AROUND 99% c. Stick it in a piece of normal fiber optic cable, excluding the cool stuff in R&D you read about, and we cut it down to AROUND 66% c. This happens because (without getting too complicated) the more 'stuff' that light has to pass through, the slower it goes. While space is pretty empty and air is pretty thin, fiber optic is pretty solid.

Well, what about copper? We put signals in copper all the time. Based on some high-level theories, copper should move electrical signals VERY close to the speed of light. In reality, factoring in insulation, attenuation, and other things, coaxial cable can transfer AROUND 80% c, and twisted pair AROUND 64% c. Wait… coaxial cable is faster than twisted pair? Didn't we move AWAY from coaxial in data networks years and years ago? Yep, 10base5 and 10base2 are dead; however, this was not done for speed, but for cost reduction. Nevertheless, coax has a grandchild in the datacenters. It is common to use a twinax direct attach cable with molded SFPs, "twinax" means "twinaxial," which is very similar to two coaxial cables twisted together.

Now, we'll focus on the items that we care about: fiber optics, light (electromagnetic waves) through air, and twisted pair.

Medium	% Speed of Light	Math	Resultant Speed
Air	.99% c	186,000 * .99	184,000 m/s
Fiber Optic	.66% c	186,000 * .66	123,000 m/s
Twisted Pair	.64% c	186,000 * .64	119,000 m/s

Before we go into individual mediums, let's hit on one additional simple topic – time – to make sure we're on the same page. You know how long a second is. There are 86,400 of them in a day, and you probably regret what you did with about 86,000 of them. In computer terms, however, that second is FOREVER. In computers, we normally – without even thinking about it – measure everything in milliseconds.

1 second (s)
Or
1,000 milliseconds (ms)
Or
1,000,000 microseconds (µs)
Or
1,000,000,000 nanoseconds (ns)

With the millisecond being our golden yardstick, let's look at our earlier chart again:

Medium	% Speed of Light	Resultant Speed	Math	Speed–Miles per Millisecond
Air	.99% c	184,000 m/s	184,000 * 5280	184 m/ms
Fiber Optic	.66% c	123,000 m/s	123,000 * 5280	123 m/ms
Twisted Pair	.64% c	119,000 m/s	119,000 * 5280	119 m/ms

Wow, the world just got a lot smaller. In fiber optics, the furthest distance that I can go within one millisecond is 123 miles… but wait, it gets worse. Almost everything that we do in modern IT requires a message AND a response. Even UDP video streaming generally starts out with a TCP connection to establish it. A round trip means that the longest distance that I can go and keep one millisecond is 61.5 miles. Unfortunately, fiber does not run in straight lines. It weaves around mountains, under rivers, dodges roads, and takes a much longer path than we expect. Traveling that 61.5 miles in a fiber cable might only be 50 miles (or less) in the real world.

Things are even bleaker for our other contestants. Air has the fastest transfer speed, which is awesome, but there's a problem. I either have to have a 'line of sight' to bounce microwaves or RF to the next location, or use satellites. Line of sight has a whole slew of its own issues; for example, the world not being flat, storms, new constructions, growing trees, apache helicopters, and all kinds of other things that can ruin your view. Plus, keeping a tight signal beam is difficult over long distances, as the air starts to disperse the signal.

The further away you want to go, the more you have to crank up the power to compensate. This causes its own practicality concerns, as eventually you start roasting birds. Medium Earth Orbit communications satellites are AROUND 12,000 miles from the earth's surface (we'll go with those versus geosynchronous satellites, as geosynchronous are up twice as high). This means I instantly add 24,000 (round trip) miles to my communications paths, which seems... inefficient.

Next, there's poor old twisted pair. The maximum distance on a single transfer medium is 330 feet, which means I have to put a repeater inline. This adds another 330 feet. By the time any appreciable distance has been run, I've added so many repeaters into the mix that my original signal has probably gotten lost, since each repeater amplifies whatever it gets, noise and all.

Let's look at another chart. Say we want to go from Nashville to Kansas City, AROUND 500 miles apart.

Medium	Round Trip Distance (Miles)	Math	Resultant Latency
Satellite	500 + (12,000 * 2) = 24,500	24,500 miles / 184 m/ms	133 ms
Fiber	500 * 2 = 1,000	1,000 miles / 123 m/ms	8.13 ms
Twisted Pair	500	500 miles * 5280 feet / 330 feet per leg	EIGHT THOUSAND REPEATERS.

133 milliseconds... that's not awesome. Have you ever noticed the lag when someone on TV is talking to a foreign correspondent in another country? It's all done through satellite. Also remember that this is mid-earth orbit; geosynchronous would be more like 266 milliseconds.

8.13 milliseconds for 500 miles round trip, though, isn't bad at all. Why then, when I log into my server in Nashville and ping Kansas over our private MPLS, does it look like this?

```
C:\Users\brent.earls>ping Kansas

Pinging Kansas [10.32.14.1] with 32 bytes of data:
Reply from 10.32.14.1: bytes=32 time=43ms TTL=249
Reply from 10.32.14.1: bytes=32 time=43ms TTL=249
Reply from 10.32.14.1: bytes=32 time=44ms TTL=249
Reply from 10.32.14.1: bytes=32 time=57ms TTL=249

Ping statistics for 10.32.14.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 43ms, Maximum = 57ms, Average = 46ms
```

There are a few reasons.

1) You never get a direct path. It wouldn't be very cost effective for service providers to run individual connections for every customer. Instead, you run to your local POP, which runs to a hub, which goes to another hub (or 12), and then makes its way to the final destination. Depending on how major the two locations are that you're trying to get between, you may be left going double the distance.
2) Networking equipment. Networking equipment adds latency, especially if it doesn't happen on ASIC and has to hit CPU. There are at least six devices inline before my ping even leaves the datacenter, much less when it traverses halfway across the country, hitting more devices at every hub.
3) Servers. The server at both ends has to actually process the ping. While this doesn't take long, it has to go through the whole software stack.

So if twisted pair is unreasonable for long distances, why mention it at all? That all comes down to your datacenter usage. I've frequently had this conversation with people doing switch inter-connections in a datacenter:

"Why are you using sfps and fiber? Those switches are only 10 feet away."
"Fiber is faster."
"They're both 1Gbit, and they're 10 feet away."
"Yeah, but fiber is faster."

Let's see what the nominal difference in fiber and Ethernet is, assuming we're going 10 feet.

Medium	% Speed of Light	Miles per Millisecond	Math	Feet per Millisecond
Fiber	.66% c	123 m/ms	123 * 5280 ft/mile	649440 f/ms
Twisted Pair	.64% c	119 m/ms	119 * 5280 ft/mile	628320 f/ms

Medium	Round Trip Distance	Math	Resultant Latency
Fiber	20 ft	20 / 649,440 fp/ms	.0000308 ms
Twisted Pair	20 ft	20 / 628,320 fp/ms	.0000318 ms

Remember that chart earlier of millisecond/microsecond/nanosecond?

That's 30.8 nanoseconds versus 31.8 nanoseconds, which is a difference of 1 one billionth (yes, billionth with a 'b') of a second. Well, let's assume we're 300 feet away, right at the limit of copper. What's our difference then?

Medium	Round Trip Distance	Math	Resultant Latency
Fiber	600 ft	600 / 649,440 fp/ms	.000923 ms
Twisted Pair	600 ft	600 / 628,320 fp/ms	.000954 ms

Well there we go. A difference of .00031 milliseconds, or 310 billionths of a second. At a normalized 1 gigabit transmission speed, that's a head start of 310 bits of data (39 bytes) for fiber versus copper. At 10 gigabit, it jumps to 3100 bits (388 bytes) of a head start. Also remember that this is only accounting for pure physics latency. The fiber connection has to be converted from electricity to light on the source, back on the destination, then back to respond, then be converted back once again on the source side. That's four medium conversions, each of which can take between two and ten MICROseconds. So, fiber wins the sprint by 310 billionths of a second, but loses the race by between eight and 40 millionths of a second.

Why go through this whole exercise? It was simply to be able to end up with this: if you're within the reach of copper and you can get both copper and fiber at the same speed, the cheaper and more physically resilient connection (no minimum bend radius; doesn't get dirty ends, etc.) is copper. Of course, keep in mind if you're running the copper through an unfriendly environment (interference, crosstalk, etc.), then there might be a good reason to run fiber.

Network:

Basics and history

Ah, networking. You have one job: push packets from here to there. Or, wait, actually, you also have to make sure they get there... oh, and make sure these packets can't go over there... oh, and make sure if there's a catastrophic failure and the city of Chicago disappears, you are online again within 60 seconds... oh, and we need you to take care of our phones... and for the love of all that's holy, STOP PEOPLE FROM LOOKING AT PORN.

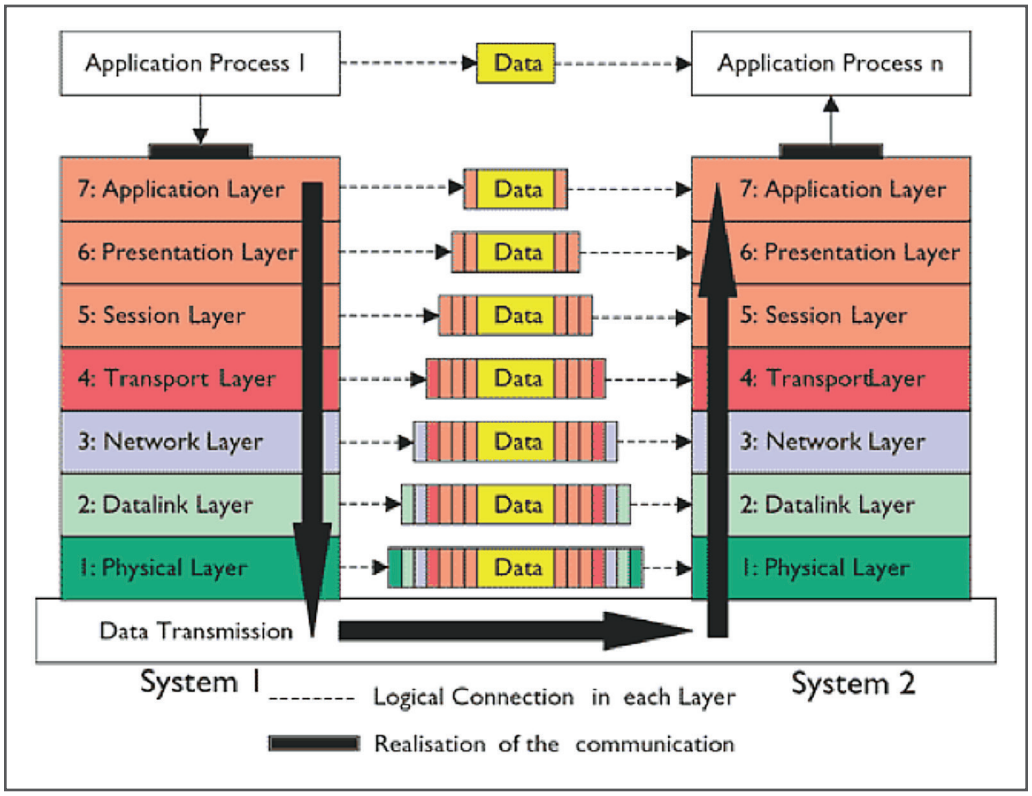
In short, networking does a lot, which is pretty impressive considering the whole thing runs on technology from the 1980s. No, really, that awesome Cisco Nexus 7000 or ASR 9000 you just purchased? Yeah, that runs on essentially the same Ethernet RFC from 1982 and TCP/IP protocol from 1981. Networking has shown a remarkable ability to continue to achieve more and more responsibility, going faster and faster, without ever actually redesigning its core functionality. Remember CSMA/CD, that protocol that helped you prevent/recover from collisions on 10base5? Yep, that's still there, and still has to be accommodated for on those modern switches/routers. Why did we just take that jaunt down responsibility/memory lane? Simply to understand how the old technology, while functional in its job, can be a part in causing latency problems.

Speaking of old legacy features, let's talk about one of the most painful holdovers – duplex – before we even get into the high levels of networking.

It's hard to fathom just how many problems in networking from catastrophic performance issues and odd behavior came from simply having a single interface in an organization set to half duplex. "Everything works great... until we start uploading this file, then the whole connection goes down!" Remember, half duplex means you can only be sending or receiving at one time. If you're on the receiving side when someone gets priority for sending, you're forced to sit and wait until they're done before you get to continue, adding the latency of waiting, being buffered, or potentially having to retransmit, depending on duration. This can absolutely kill performance.

Luckily, it's now impossible to get a port set to half duplex – at least with 1 Gbit. However, for all the 100 Mbit ports that are still floating in your organization, pay careful attention to duplex. A lot of networking people take the global approach: "Well, I'm just going to set everything to gigabit full duplex," which is great until you plug in something can only do 100 full (yep, there are still several devices out there, even in datacenters that, at some point, link at 100) and you have to troubleshoot why it doesn't even get a link light. Half duplex negotiation can cause catastrophic networking latency problems; luckily, they're often severe enough to be investigated.

Ethernet + TCP/IP has a lot of inherent overhead. Think of how networking works at a simple high level method. Rather than vocalize this, I'll simply point to a diagram that we all saw no less than 20 different times in school/studying:



http://edugeeks.in/osi-model-computer-awareness-material-for-bank-exams/

That's a 'crap-load' (technical term) of headers on data, not to mention several of these layers are so intertwined/overlapped that it's hard to differentiate between them. The datalink layer controls hardware addressing, but the network layer concerns software addressing, which means we have to address the same thing twice. Layers 4-6 become a complete disaster for normal people to try to differentiate exactly what does what. In short, while it's a very complete protocol stack, it's a bit large.

There have been GREAT strides in making these layers faster over the years. For example, at layer two, we've got CAMs looking up MAC addresses at line speed, something that RAM (which, in servers, we think of as being ungodly fast) couldn't match. We have cut-through rather than store and forward, which starts to push a packet out before we've even received the whole thing. At layer 3, we have TCAM to assist with routing address lookups; again, at a ridiculous speed.

However, it's not all good news. Some protocols have only had limited improvement; for example, look at spanning tree, which went from AROUND 50 seconds down to AROUND one to 10 seconds with RSTP. Seconds... weren't we just measuring things in billionths of a second a few paragraphs ago? Luckily, spanning tree doesn't have to happen all that frequently, but when it does, it can cause massive problems. Additionally, if your spanning tree isn't properly designed, implemented, and tuned, it's possible to end up with very nasty and inefficient paths between two devices that are directly connected. Lest we forget, barring configuration and all priorities being equal, the STP root will be the lowest MAC address (aka: oldest equipment).

At layer 3, TCAMs CAN make routing/ACLs/QoS awesome, as long as we have enough TCAM space. Once we turn on something like Policy Based Routing (PBR is my whipping boy because it's caused me so much grief over the years) and other features, we can easily exceed TCAM space and have to start offloading things to the CPU of the router/switch rather than ASIC. You may or may not know this, but switches/routers have some of the oldest and slowest CISC-based CPUs you'll find in your entire organization. We're talking six years or older Celerons here.
Policy Based Routing can take a router that's running at 90 Mbit/s and take it down to 5 or 6 Mbit/s (depending on a copious number of factors, obviously). Improperly sized/implemented Policy Based Routing is about as bad for your network as PBR beer tastes. And of course, when we have a smaller pipe, we have to queue items in line and process them as we get to them, rather than pushing them through quickly. This quickly takes us to our next item...

Uplinks
Uplinks are the backbone (literally) of any network, but people frequently lose track of something simple when it comes to uplinks: overcommitment ratio. Suppose you have four 48-port gig switches stacked. Those switches connect to each other in a stack anywhere from 32 to 480 Gbit/s, depending on models and cables, etc. If we then link those switches back to our redundant cores, each with a 1 Gbit connection, most probably (depending on protocols) spanning tree will down one, so we have 192 1Gb ports trying to connect to our core through 1 Gbit. This means all that each port would have to push is 5 Mbit/s (.6 Megabytes) in order for us to exceed and saturate our 1 Gbit uplink and everyone would then have to wait their turn to get across the pipe. It may also lead to all kinds of nasty retransmits, buffering, QoS shunting, traffic shaping, and other things on the switch that further restrict how quickly our data traverses. Needless to say, this is not awesome. This is especially important to keep in mind with most modern datacenter switches stacking over 10/40Gbit Ethernet ports rather than actual stacking ports. Always keep in mind oversubscription ratios.

One place where this is becoming very important in recent times is wireless. If you have a wireless environment where the APs plug into your access switches (as is perfectly normal), remember that most APs now come with gigabit interfaces. Also remember that modern AC wireless is "gigabit wireless." You'll never get a gigabit out of it in the real world, but even if your clients are only pushing a couple hundred megabits, that can quickly add up to a substantial amount of traffic coming into that wireless AP, which then has to go through your uplinks back to your distribution/core networking. As long as we're on wireless, the largest source of latency in wireless networks is the wireless signal itself and its over-proliferation. Have you ever been to a really big conference or stadium and had five bars of signal, but can't get pings even to your default gateway?

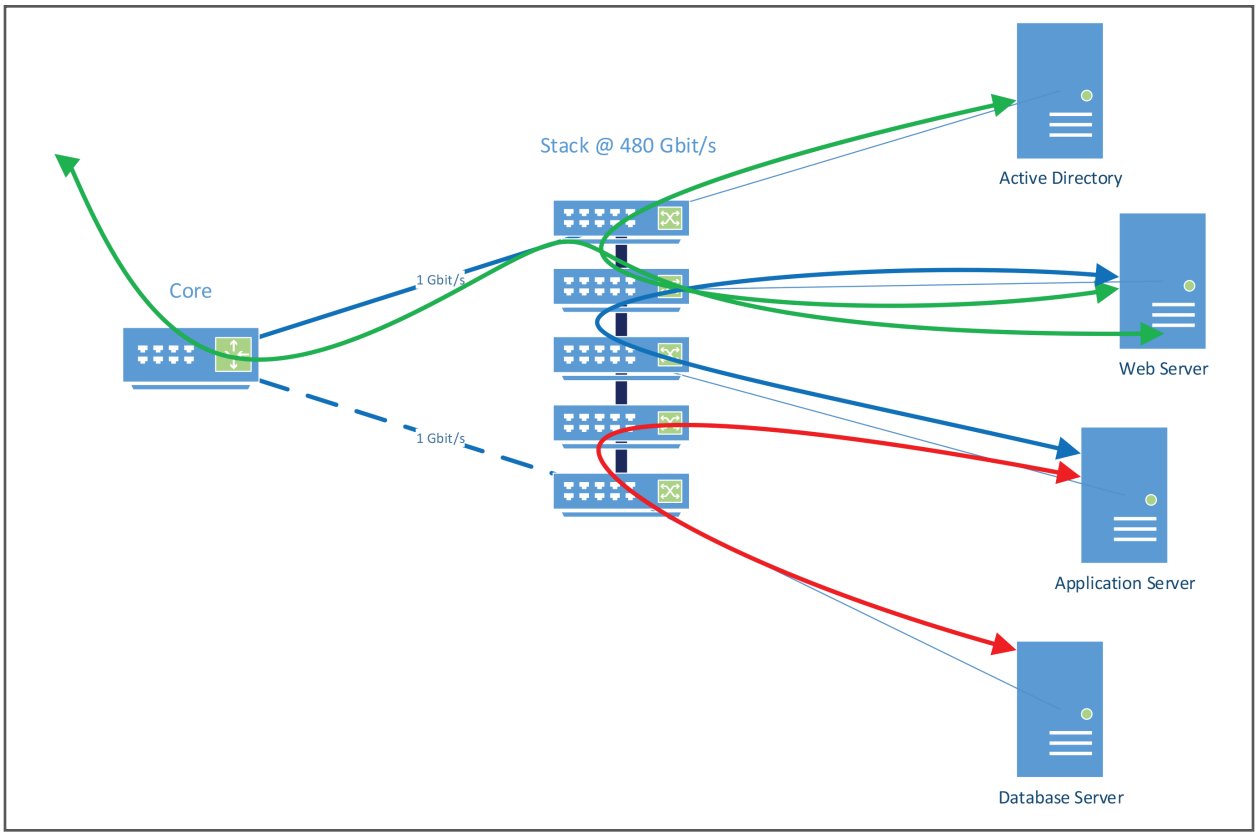
That is often a problem of over-utilization. Wireless is still a broadcast-based technology. If a couple hundred clients are all trying to get on a single AP, that AP runs into issues just trying to get all of those various signals in and out one by one, which leads to latency on traffic, if traffic even happens at all. Often, attempts are made to compensate for this by putting a very large number of APs in an area so each one only has to handle a smaller subset, but that tends to lead to crosstalk and interference amongst the APs themselves.

This is experienced, to a lesser degree, in a lot of office buildings where other businesses are on all sides, all with their own wireless that are all bleeding over each other. Remember, there is no easy way to contain wireless within an office space (unless you build your whole office to be a giant Faraday cage).

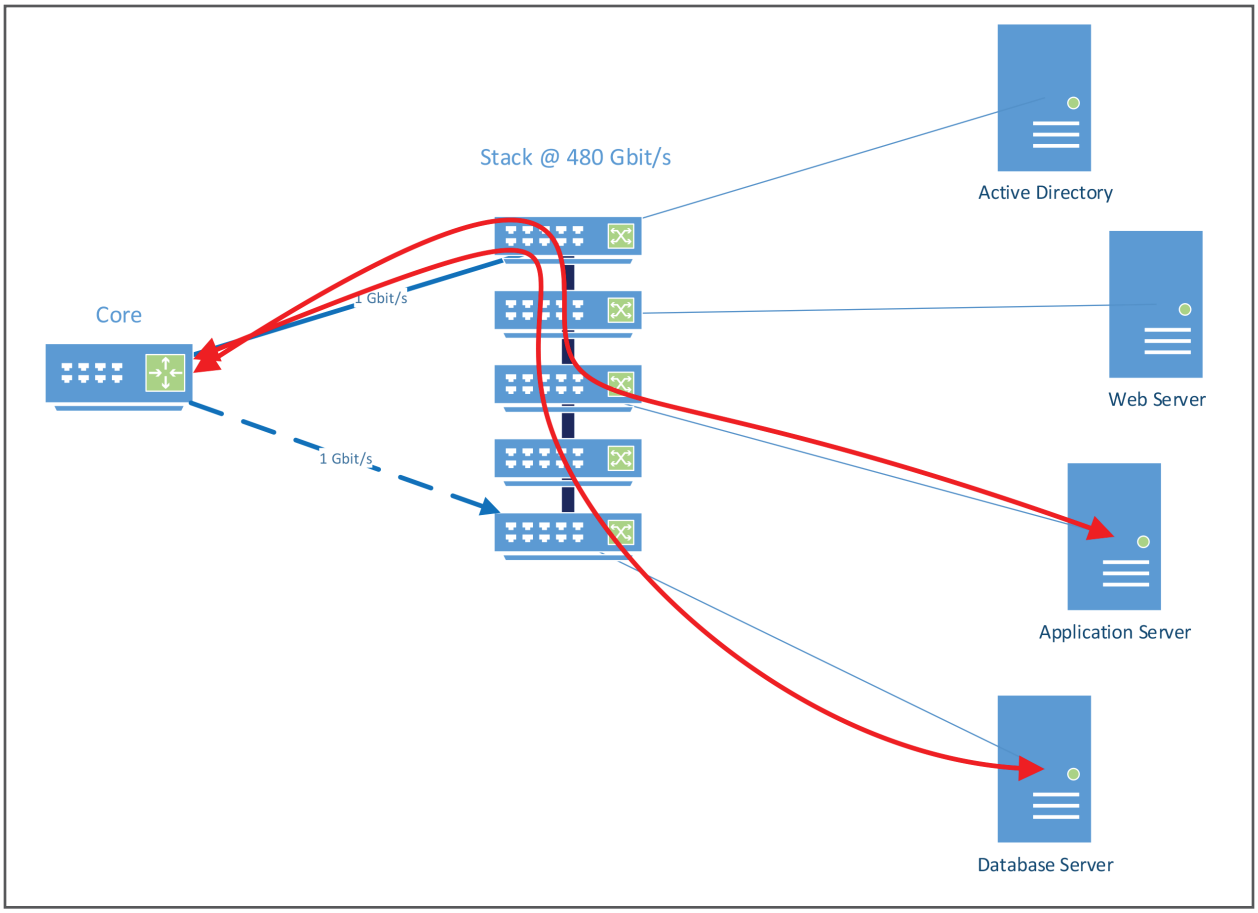
Routing

Another major latency point in datacenters is routing. Routing in and of itself isn't a bad thing; however, if it's implemented incorrectly, it could cause a massive amount of woes due to unnecessary hops. For example, say someone decides to VLAN their environment but keep all routing on their core. They choose to segregate based on the functional role of the server: SQL VLAN, Application VLAN, Web VLAN, Active Directory (AD) VLAN, Exchange VLAN, etc. They stack switches with two uplinks to their core (STP takes one down). In the previous design, if the web server got a request, it came in from the core, talked directly to AD for authentication on the same stack at stacking speeds, then went to the application server, which went to SQL and then back up, all at stack speeds. Now, the web server gets a request, has to traverse to the nearest uplink to the core (potentially across multiple switches), go across that uplink, hit the core, get routed, go back down the uplink to the stack, traverse switches and get a return, which then routes that same path to go to the application server, which then routes the same path to go to the database and back.

Here is an oversimplified diagram for explanation's sake. In the old way, with servers in the same VLAN, they hit the stack and traverse quickly to the other servers:



But the new way isn't so easy. I highlighted the worst case scenario here with the database to application server, mostly because the diagram quickly became a jumbled mess with all of the links shown:



The poor application server has to go to the switch stack, jump two switch stacking cables, go to the core over 1 Gbit, get processed, and respond from the core to the stack, which traverses four stacked switches to finally return to the database server... which then reverses communication back. It's a much more roundabout way of communicating, AND rather than keeping it on the super-fast stacking links, it now hits normal 1 Gbit links. The latency between stacked switches should be fairly negligible, but it all adds up. More importantly, we're now putting the load for all of these server communications on that 1 Gbit connection, adding to latency while then throwing the router's latency into this whole mix. Let's not forget about the potential bandwidth bottleneck of the router, and the router's ability to process all the extra traffic. If either the pipe fills up or the router gets overloaded, additional latency starts hitting our traversal paths. Far from an optimal scenario, but not an uncommon one. Routing isn't bad, but you HAVE to think through your design and trace where the communication paths happen.

Internet

Internet is the most frequent pipe that is oversaturated and has latency. In America at least (our internet bandwidth isn't great), it's a very common problem to run into. Are people complaining about your wireless being slow at the big public event you're hosting? Maybe it's actually because the internet pipe is overloaded, not the actual wireless. Users complaining of slow desktops?

Maybe it's because they're trying to use a web app and there isn't enough bandwidth for it to actually load. Internet bandwidth is very frequently a bottleneck that can increase latency elsewhere (if you take one lesson from this paper, let it be that if a pipe is full, latency increases). Also keep in mind, that the speed test you run from your ISP is normally not leaving their network. Those insane latencies and bandwidth numbers aren't representative of the real world. For real world numbers, find a free bandwidth test site, and then pick the furthest location away from your datacenter where you have users accessing your information. That's the latency you need to be able to accommodate in your overall design.

If you ever wonder how a whole team of networking people seem to be able to keep busy all the time, this is why. It's complicated. They aren't simply playing wire monkey plugging in a lot of equipment (that's what interns are for), they're working on complex designs to try to eliminate, or at least compensate for, as many of these potential pitfalls as they can. If they aren't working on new designs, they're often trying to figure out how to force someone else's bad design to do what they need, all while working with antiquated protocols. Not even the almighty Cloud can help the networking folks; they still have to be able to get your data up there (more on that later).

Servers

Hardware Concerns

Once we get to our servers, there are plenty of additional places where we could encounter latency. Modern CPUs, RAM, and PCI-E busses are obscenely fast compared to what we had a few years ago, but there are still some key places where slowdowns can happen. To find these, let's trace the path of information as it comes into a server and gets processed.

Information comes into a NIC, travels down the PCI-E links to the CPU, and tells it that there is an interrupt process that needs to be processed. The CPU then clears some cycles to be able to accept this NIC traffic. The actual PCI-E link to the CPU is one of the first potential bandwidth points. A few years ago, the PCI-E controller was pulled into the CPU itself, which means that now, rather than having all of that go to the Northbridge and then be piped back to the CPU over a smaller highly contentious pipe, the PCI-E card has a direct link to the CPU itself without restrictions. This is great, and it massively reduces latency and increased throughput... as long as the card is in the correct PCI-E slot. The potential problem with this new direct-to-CPU design is that now, there are only so many connections to go around to all the PCI-E slots, and they can't be shared (barring expensive PCI-E switches).

Because of this, you run into some modern motherboards that have x8 physical slots that only have x4 electrical connectivity, while some x8 slots have the full x8 connectivity. If you put a card that has an x8 slot in one of the physical x8 but electrical x4 slots, you could limit its performance greatly, as it now doesn't have the full bandwidth back to the CPU. This, again, leads to the potential for bandwidth restrictions, which leads to latency. The fix for this is simple: consult the manual (I can hear the collective sigh through the internet) and see what the manufacturer says the appropriate slots are for the highest performance.

Another important configuration with regards to all of this is your RAID card interface. Remember, if you're using internal storage, it's on a RAID card that is subject to all of these same PCI-E limitations. If you put 32 SSDs behind a single x4 PCI-E slot, you aren't going to get the performance you expect. Each PCI-E 3.0 lane supports 985 MB/s, so that x4 slot can sport a total of 3.94 GB/s, which is not a lot if you're dealing with a load of SSDs.

Once we get to the CPU, we have more potential latency concerns. Modern CPUs have also pulled the memory controller directly into the CPU, creating what is called NUMA (non-uniform memory access) nodes. This means that the bandwidth between the CPU and the memory has skyrocketed up to AROUND 68 GB/s (over 500 Gbit/s), but it also causes a potential problem. In a multi-CPU system, there has to be a way to let each CPU talk to the other memory. In the Intel world, this is through a Quick Path Interconnect (QPI) between the two. This is a VERY fast connection with up to 9.6 billion transfers a second (GTs), which is fast (AROUND 38 GB/s), but it is still not as fast as the local memory access we were just talking about. Additionally, it adds some additional hops for our data if we need to access memory from the other CPU, as it has to hit the first CPU, traverse the QPI, hit memory, and then repeat the process back to the source. This can result in additional latency; however, with the connection between the CPUs being slower than the memory access, if a lot of data needs to come from the other CPU's RAM, it can saturate bandwidth and cause additional latency waiting for the line to clear between the two. This can be fixed by properly configuring your RAM (back to RTFM) as per manufacturer instructions and properly tuning your OS (mentioned later). Additionally, with RAM keep in mind that how the DIMMs are populated and which DIMMs are used can greatly impact the overall memory performance. For example: even when using the fastest DIMMs available, if all slots in a server are populated, it may degrade memory performance.

Operating System Concerns

Operating systems, specifically hypervisors, make a massive difference into the way in which a server performs and, subsequently, on the latency involved. The years of "you can't virtualize that" are quickly fading, mostly due to ever decreasing latency numbers. More CPUs, more RAM, better storage access, and better software intermediaries to the hardware all helped to decrease latency, which has massively increased virtualize-ability. One of the most important places this all comes into play is with getting your VMs onto the CPU.

While Hyper-V can schedule individual CPU cores independently, VMware has a 'relaxed co-scheduler.' Therefore, if you have a VM that has a lot of cores, at some point they all have to be scheduled on the CPU at the same time. In order to do this, the CPU may have to kick other VMs off the CPU to free up room, which can lead to a state called CPU Ready that causes latency.

This latency is often measured in our familiar millisecond metric. Remember, however, that these are CPU cycles we're scheduling. CPU timing is often measured in billionths of a second, and a couple of milliseconds can have a profound performance effect.

Multiple times, customers have greatly INCREASED their overall virtual environment performance by DECREASING the number of virtual CPUs. This allows for the VMs to get more shared access to the physical cores and not have to fight over them. To avoid this, pay attention to overcommitment ratios of your virtual CPUs to physical CPU cores. Four to one is considered a fairly healthy ratio. Also, always try to keep your VMs sized under the number of cores on a single socket, or make sure that vNUMA is enabled on that VM so it can properly have cores scheduled closest to the memory. If this isn't done, the vCPUs can be scheduled on the wrong NUMA node and cause more unnecessary data access hops, and therefore produce latency as discussed above.

Memory is another key place for latency for VMs. Normally, memory performance for VMs isn't bad, but if they aren't properly designed, the VM can start swapping memory to disk. This isn't the controlled paging that Windows or Linux does on a daily basis, this is the hypervisor randomly pulling pages out of the VMs' memory and sticking them in the paging location. Even if that paging location is an SSD, it's still an order of magnitude slower than RAM. Think in terms of SQL thinking something is in RAM, going to access it, and getting a response back 300 times more slowly than it expected -- it won't be happy. This can be prevented by not overcommitting RAM on the hosts, as well as making sure there are no artificial limits set in the VM configuration.

Virtual networking is, comically, one area where latency isn't normally a problem. There are a couple of key points, though. Be cognizant of how overcommitted the virtual uplinks are to the rest of the environment. Running 30 or 40 VMs on two 1 Gbit uplinks can run fine for many customers for a long time, but keep in mind that if those VMs start trying to actually use their full bandwidth, they can essentially drown the other VMs and cause latency. Remember the earlier network uplink overcommitment ratio? It applies to virtual switches as well. Also, keep VM access patterns in mind when setting up affinity rules in your virtual environment. If your web, application, and database server are all on the same host, they can potentially talk to each other without ever having to leave the host itself, giving high bandwidth and low latency communications. If one of those servers moves to another physical host, it will now have to traverse one or more different switches in order to communicate, dealing with their latency and more potential bottlenecks. VMware NSX has the ability to potentially help some of these performance problems if properly implemented.

Pay careful attention to keeping integration services/VMware tools up to date. That's the primary way in which a VM talks to the hypervisor. Outdated integration services/VMware tools can cause all kinds of issues, including bizarre latency issues or simply generic failures.

Finally, if you're still running workloads directly on physical servers, the most important detail is to always keep the firmware/BIOS up to date, as well as all drivers. Also, pay careful attention to any recommendations for BIOS settings from your application vendor. Certain applications don't function well with hyper threading, turbo boost, or C states. Those applications can get more latent or perform poorly in general while the feature is on, as the software essentially gets confused about the different, potentially uneven speed results it gets back. Keep in mind that is not a general recommendation, as most workloads benefit from or at least aren't harmed by these settings.

General

Aside from interconnects, storage is the other big place you hear a lot of conversation about latency. Everyone is always very excited to talk about their newest storage array that can deliver one million (insert Doctor Evil joke here) IOPS, or complete the SPC-1 benchmark at only two milliseconds of latency, but what do all of these things mean? Is one better than the other? Why is everyone so excited about all of this, and how does it really relate?

Theoretically, if I can do one million IOPS, that means each operation is only taking one microsecond (a millionth of a second), so something that can work that fast will never add any appreciable latency, right? This is not necessarily true. Before you tune this out, thinking, "I don't care about SANs, I run things on local disks," most all of this still applies.

The first key thing to start with this discussion is that IOPS on their own don't mean anything at all. IOPS literally just means "Input Output Operations Per Second". It doesn't have any criteria whatsoever around what size or type of operations those are. For example, a storage company who shall remain nameless (because who am I to throw stones?) did a press release for one billion IOPS. Yep, with a b, BILLION. If you look at the press release, though (okay, it was FusionIO and clearly I have a rock in hand), they were pushing 64 BYTE blocks. Not KB, BYTE. That's AROUND 60 GB/s. This is nothing at all to sneeze at, and neither is one billion IOPS (I'm not trying to discredit that), since no one writes in 64 byte blocks. If they were using something more reasonable, like 8k blocks, it would have been AROUND 8 TB per second; 128 times as much data. Or, if they had done the same bandwidth at 8k blocks, it would have only been around 8 million IOPS (again, still nothing to sneeze at). The point of this endeavor was to acknowledge that the key metric that we use to talk about storage speed (IOPS) is completely useless on its own, as are latency and bandwidth, or front-end benchmarks without back-end product descriptions. We have to know the whole story to get any real, useful information out of benchmarks.

Another noteworthy detail is that everyone quotes numbers as averages. If I have 500 IOPS at three ms latency and one IOP at 1000 ms, the average is still 4.99 ms, but whoever had to wait a full second for their IOP is going to be angry. You would like to think that if I push one million IOPS that I can normalize that down with some math and figure out the latency of each IOP, but it's not the case. Often, the way these extremely high IOP numbers are perpetuated is by having really deep IOP queues. Those allow the storage array's controllers to cherry pick the IOPS that it can most quickly satisfy and get to without expending a lot of work (cache hits, localized access to other data it's already retrieving), while leaving other IOs languishing. These deep queues can also lead to IOs sitting around waiting in queue on the host before they even get to queues on the storage controller. It's possible to have latencies of 20 to 40 ms while still pushing insanely high numbers. Once again, this can falsely inflate performance numbers. There is no one magic metric that allows us to quantify the speed of storage.

On our earlier example, let's say we have a storage vendor quote the SPC-1 benchmark on a single chassis at 252,000 IOPS and 2ms of latency. That may be far more impressive than one million IOPS, because the SPC-1 benchmark is an industry standard, controlled benchmark that represents real world performance, and we see that it has low latency. You can't stop there though, because what if there is a whole rack of equipment behind it costing a million dollars? Maybe not so impressive again. Oh? It's one 3U chassis that goes for less than $81,000... Okay, now that's impressive, tell me more! Sadly, not everyone plays nicely and gets their systems benchmarked by a third-party impartial judge. So, let's go into how storage works and discuss all the wonderful places where latency is added, either directly or indirectly, and why storage has such a buzz.

Disks

Let's start simple with disk latencies. When it comes to physical disks, latency is the most noteworthy speed-limiting factor. Two disk latencies come into play with spinning disks: rotational latency and seek latency. Rotational latency is exactly what it sounds like; the latency coming from waiting for a disk to revolve around in order to get the heads near the part of the disk that they need to get data from. On a modern 2.5" 10k drive from a well-known manufacturer, this listed as average of 2.0 ms on a 15k, 2.9 ms on a 10k and, 4.1 ms on a 7.2k. This number lines up about how you would expect, because a 7.2k drive spins at about 70 percent the speed of a 10k drive. Nothing unexpected there.

The second number is the seek time, which comes after the head gets around to the correct part of the disk. This number determines how long it actually takes to locate the data that needs to be read/written to. Looking at the same spec pages as above, 15k drives average around 2.9 ms, 10k drives average 3.3ms, while 7.2k average quite a bit higher, around 8.5 ms. It's also worth noticing that, while the gap from 7.2k to 10k is rather huge on seek times, it's not large going from 10k to 15k. All that the faster rotation really helps with is reducing the rotational latency, but it does so at quite a cost of increased power and actual dollar bills. The big cost and power gap without a big performance gap between 10k and 15k is why you don't see much new development in 15k drives, while 10k keep getting larger and larger. The key to all of these numbers, though, is that wonderful IT term: average.

SAN manufacturers have always enjoyed short stroking their stuff in order to increase their responsiveness (keep your mind out of the gutter). Reading/writing to the outer tracks of the drive allows the drive to be much faster as it lowers seek times drastically by not forcing the head to swing over the full geography of the disk. The problem is obvious, though: you don't get the full space of the drive. Once the inner tracks of the drive have to be used, performance starts to deteriorate as more and more seeking is necessary, increasing overall average seek times. SAS, NL-SAS and SATA all have different technologies built into them to try and optimize the pattern by which drives access data. These technologies mean that if there are two different read requests that come from the same general area, they can get processed at the same time, even if they weren't received in that order. This can help overcome some latency by processing dozens of IOs from one area quickly; however, it may mean leaving some IO from a different locality hanging out for a long time. Because of these latencies, random reads are the worst things for traditional disks, while sequential reads and writes are handled admirably, and random writes are okay WITH a good cache.

Always pay attention to the term AVERAGE when looking at drives.

SSDs

Then, there are SSDs. Yep, they're so different that I don't even talk about them in the same segment. SSDs offer insanely low latencies that can be measured in microseconds rather than milliseconds. SSDs can offer average random reads with latencies between .1 and 1 ms, often spiking up to averages in the 3-12 ms range. Again, those are reads. Writes are where SSDs have a lot of technical struggles that are mostly overcome with some creative software. MLC and eMLC drives, which are the majority of all SSD, can read in small increments called pages, which are often 4k. They can only write in a group of pages called a block, which is around 32 pages (sometimes more or less depending on how the chip is made). However, don't confuse this SSD block with the block size you formatted your hard drive with, as they're completely different. Practically, what this means is this: if I need to write 1k to update a single 4k page, I actually have to read the whole 32 pages (128k), make our change in memory, zero the block, and then write the whole block back again. That's less than optimal, and can lead to a lot of (you guessed it) latency.

This write amplification is why SSDs use TRIM and garbage collection to try to prevent the need for rewriting to the same place. When new writes need to happen, the old data will be read and then written to a pre-zeroed area of the SSD. This all works well, until an SSD starts to get old and wear out. At that point, the read/update/erase/rewrite process can start to happen, which can increase their write latency (and decrease performance) by a factor of 10 or potentially even more. This can suddenly start making SSDs slower than spinning disks for writes, which can really put a damper on your database. If you ever used first-gen SSDs before TRIM in your laptop, you probably felt this pain. One of the biggest differentiators in the last few generations of SSDs is their ability to mitigate read/update/erase/rewrite processes and to survive for longer periods of time. This is why proper storage array design is so important, as one must accommodate for these various factors and their influence. Due to these concerns, SSDs are GREAT at random reads, sequential reads, and sequential writes (just don't burn them out with non-stop writes every day), but they tend to struggle with random writes in certain situations.

RAID

Watch out, it's a RAID!! I think we're all infinitely familiar with RAID at this point. If not, we have a good primer on our website. Assuming you are familiar with RAID, let's go over a quick run of writes, another place where you can get latency. Remember, on RAID 1/10 you get half as many writes as you get reads on the same RAID set. On RAID 5, if you aren't doing full stripe writes, you get 25 percent as many writes as you get reads, due to the read, write, parity calculation, and parity write operation. This can also impose some additional latency, as it decreases bandwidth. On RAID 6, if you aren't doing full stripe writes, you get 17 percent as many writes as you get reads. All of this is important, because if you start to get big write performance penalties, you can quickly run into a situation where you over-reach the write performance of your array. That can cause additional latency, as your IO have to be queued to wait for disk resources. Another key to think of is that a RAID set is only as fast as its slowest disk. Again, this is something we all probably remember from our primers on RAID, but it becomes a big concern when we're talking about latency.

One can't simply compensate for the speed of an individual disk by adding a whole bunch of those same disks to a RAID and expect quantity to compensate for spindle speed (I'm looking at you 7.2k disks). Each disk still has to do its own seeks and return results to the RAID controller; there are just a lot more of them doing the slow operation. The increased latency of each disk also leaves more room for the aforementioned RAID overhead hit. You've probably experienced this effect at some point in the past, when a server suddenly started performing badly and the root cause was that a disk was in the process of dying but hadn't quite died yet. The whole array was brought down to its level.

Other features and concerns.

There are a few more things to keep in mind for storage latency. All of those fancy additional features that people throw on the array, like compression and dedup? All of those features have some form of overhead, either decompressing or rehydrating (re-duping) data as it's accessed, which can add latency to the array.

Auto-tiering is another great feature that can do wonders for accelerating certain parts of workloads; however, keep in mind that you now have two completely different types of disk in the same LUN, which can lead to very different latencies for different parts of a disk and unpredictable performance. Many auto-tiering products only tier once a day, which means that if data heats up in the middle of the day, it won't be accelerated until that evening when it may no longer be needed. Ask your vendor what their auto-tiering time period is: one hour? Cool. Five seconds? Amazing. Additionally, keep in mind what the penalty will be for missing your higher tier. If I have a super-fast SSD tier that goes straight to 7.2k drives, the penalty for something not being properly tiered to SSD is huge and can destroy performance.

RAID/SAN/Host side caching has a very similar problem. RAM is very fast and can greatly decrease the latency of your system response, providing a simulated higher IO than it is actually processing; however, remember that this isn't the actual permanent storage of the data. If the cache is exceeded, your throughput/latencies start to fall back to at least what the disk supplies. This can sometimes be even worse than the root disk performance, depending on how overwhelmed the array is at that point in time. Also, if there's a situation where the read cache data is lost (controller crash, or firmware update), then the cache goes away and has to be rebuilt from scratch; which means performance can be greatly affected.

Finally, when considering performance of storage, access methods are very important. Is the IO sequential or random? We covered what each type of medium is best at above. In virtualized environments and SANs, there is essentially no such thing as sequential access anymore. The reason is that an IO stream is constantly being interrupted by other IO streams from other hypervisors, hosts, or VMs. This means that your RAID card/SAN needs to have good caching/prefetching mechanisms in order to compensate for those random IO streams and get sequential IO out to the disks (remember, even SSDs benefit from sequential writes).

User Experience

Oh, users. Working in technology would be great if it weren't for those pesky users. If all we had to do was make the little lights go blinky blinky, we'd all be able to have a fulfilling job without stress. Unfortunately, the users are who pay the bills, so we must accommodate them. There are a few different ways in which latency affects end users, and pretty much none of them are good. We can start with the end-user computing device itself. Essentially, all of the premises and topics that we've covered above apply to an average user desktop. Is their storage fast? Is their CPU overwhelmed? Is their video card in the right slot? Is their connection to their switch/Wi-Fi adequate? All of those things can greatly affect how happy a user is before we ever involve the whole infrastructure that we work with. Have a power user? Do they have an SSD in their system? How about all your bosses? The key to end-user experience is that you are essentially running on a ledger of good will that is directly proportional to latency (and a number of other factors, but those are for future articles). Every step of their process has latency in it, and everywhere you overcompensate for latency helps to absorb the latency from other levels. A person starts to feel latency as a real thing with their clicks and other things at AROUND 100 ms, which sounds like an eternity since we've been measuring so many things in nano and micro seconds, but that user latency includes EVERYTHING we've talked about, often multiple times.

Let's walk through what a few of these latency progressions may look like for a few different users: a user on site and through an MPLS, a VPN user on a remote internet connection, a user at a branch office on VDI, and a user when the services are hosted "in the cloud"

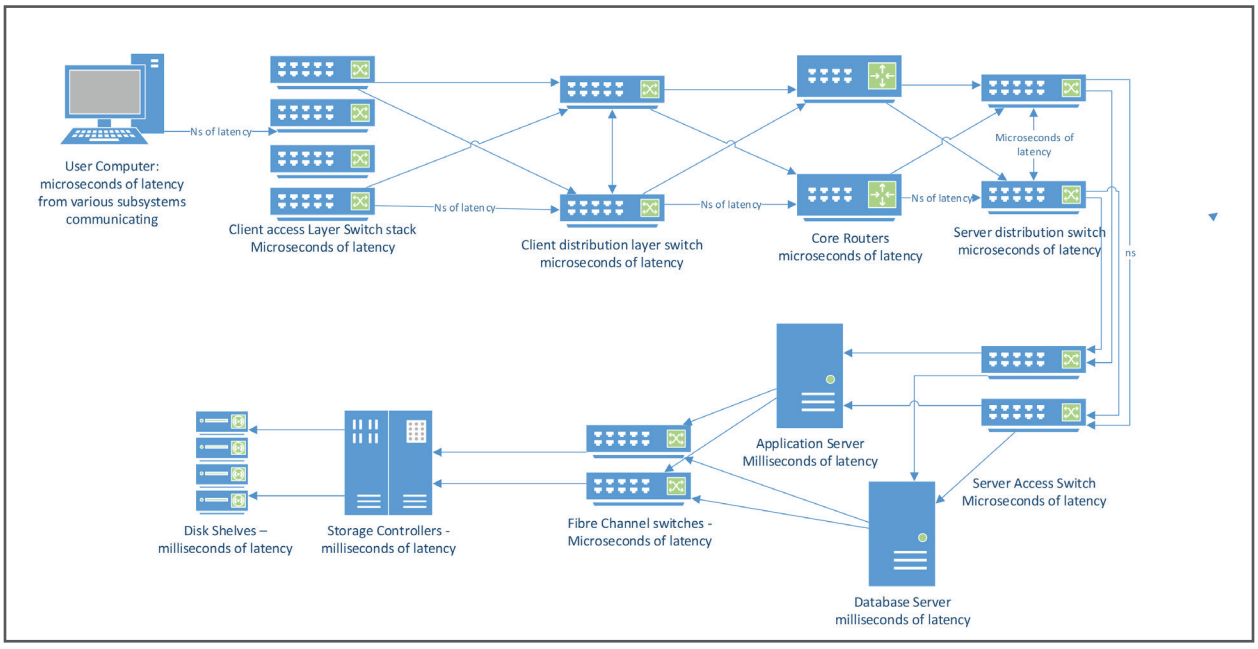
Local on-premise user:
We're going to walk through this first example in exhaustive detail. After this one, it will only be iterative. The user sitting at a desk double clicks on your 'line of business' application. The computer has already completed thousands of operations internally before it even loads anything. This shortcut now points it across the network. The computer traverses across its Ethernet cable to the wiring closet, incurring a few nanoseconds of latency before hitting the switches. It then gets switched through the stack to the uplink, and this may impose a few microseconds of latency depending on how the switch is configured. Our traffic now hits the transceiver and fiber uplink to the distribution layer, incurring latency on transceiving and fiber traversal; again, only a few micro seconds. The traffic hits our distribution layer, again getting switched through an uplink to our core. At this point, it gets routed to another network and sent down more fiber to the server access layer, where it's again switched to the correct port for the ESX host (picking up a few more microseconds of latency in the process).

Traffic traverses the virtual switch and the port group before getting down to the VM, where processing begins. The hypervisor schedules the CPUs of the VM on the physical cores of the ESX host and processes memory requests and needs from the VM, picking up a few more microseconds of latency on a properly configured system. The VM then processes the transaction and reads from its virtual hard disk, which reads from a VMDK on a VMFS datastore, which

then has to be converted to a fiber channel request, which again goes to a switch and then to a SAN, picking up another few microseconds in the process.

The storage array then has to send the IO through the hardware/OS stack, check the request at cache, go find which RAID set the data is on, and send a request down the SAS connection to that enclosure, adding a few more microseconds of latency. The disks then have to rotate and seek the data to find the necessary location, which generates a few milliseconds of latency. The response traverses back through the SAS connection, back through the HBA in the SAN, through the entire hardware stack of the controller, gets processed by its OS, sent back out the Fibre Channel HBA, traverses the Fibre Channel switch, and lands at ESX. ESX then processes the IO at the HBA and sends it to the appropriate VM, who reads it only to find out that it needs to consult the back-end database server. At this point, we've picked up a few ms of latency, but we're not even halfway there. We have to send a network request to the other database VM, which again goes through the port group, virtual switch, and down to the physical switch, which will switch it over to another ESX host uplink, which will traverse through the stack to the database VM; again, picking up a few microseconds of latency. This request then goes back through the storage stack to grab data from its database and processes it, imposing a few milliseconds of latency. This is then returned to the first VM through the VMware and physical server access layer switch, which processes the response and then sends a response back to the user.

This response reverses the earlier path, going from switch access layer to core to distribution to client access layer to client computer and gets processed as information on a screen. All of this just happened within a few milliseconds, most likely somewhere around the 30-50 ms mark, depending on how long the application and SQL processing layers take and how long the SAN takes to find the data. Notice that almost all of the latencies imposed were in microseconds, except for the storage and application layer. This is why on-premises computing can be so fast, as the only things that really slow it down are the seek times for storage and the application processing time; most notably, how many times it has to go back to storage.



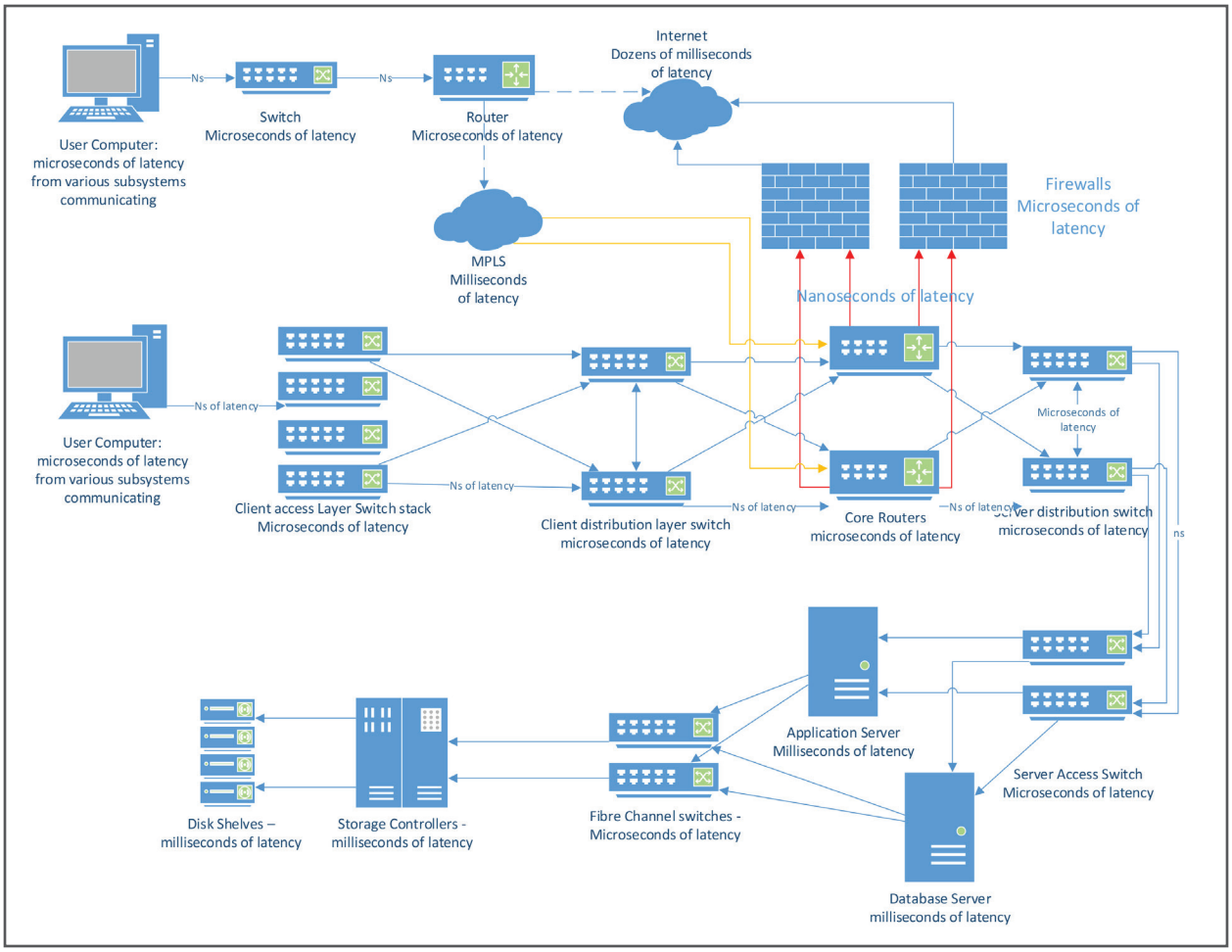
That's what leads to the key takeaways here. If the client is on premise, the actual round trip isn't a problem; even, if the client needs to talk to the application server 30 times, the latency is negligible. The "problem" is normally with the backend processing that happens between the devices. In the case above, storage was our limiting factor. This is where it's very important to know how your applications function in order to know how they will operate. Is there a lot of back and forth between your client and the server, or does the client send one request, the server do a lot of work, and then send one response? The constant chatter back and forth between clients and the server is really common when clients connect directly to databases, as well as in many older applications (true client/server). In the above situation, this wouldn't be a problem, but what if we're traversing an MPLS or long point-to-point connections?

Long Distance (point to point/MPLS/VPN)

When we add long distance latencies into our mix, we suddenly start adding latency between our processing and requesting points. In the above situation, the only latency that will be added to our round trip is the actual sending of the request and receiving an answer. This is maybe 10 to 50 milliseconds total. What if our app was one of those chatty older apps? Now, every single back and forth operation is going to keep getting added latency, which can very quickly add up to an appreciable delay and slow down our overall functionality. If we had the 40 millisecond round-trip latency mentioned before (from Nashville to Kansas City) and our application has to talk back and forth 20 times before giving the user a meaningful answer, almost a whole second of delay was just added to the overall operation. This is something that can really be felt.

What about a VPN? VPNs are more unpredictable because you have no idea how you are going to get from point A to point B, and there are no real SLAs on your connection. One time, you may connect from Nashville to Kansas City by way of St. Louis; the next time, by way of Dallas or Chicago. This unpredictability can quickly lead to large and unpredictable latencies. There's no QoS on the internet, so your VPN can add either a few milliseconds of latency or a whole lot more. Also, keep in mind that the VPN has to be processed by a VPN endpoint. That device (depending on how many hats it wears) may incur more latency as well as it processes through ACLs, IPS, and routing for the VPN endpoint. If you've ever had a user work "fine" from a branch office, and then "badly" from a VPN two miles away, these unpredictable factors can very well be the reason. Also, remember that the internet pipe can start to fill up. If the pipe is full, your connections to the server will be more latent.

Again, your mileage may vary based on how your application works. If all the processing is happening directly between the client and the database or server, this may be notable; if it isn't, it may fall under the detection line. Keep in mind as well that the user's interaction point can make a big difference here. If the user clicks in a web app or thick application, it may be able to hide a bit of the back-end latency, because things at least feel smooth from the interaction point, as their mouse and typing and such still at least gets processed quickly for them. It's only the results of things that seem to take a while. Here's where VDI makes things interesting.

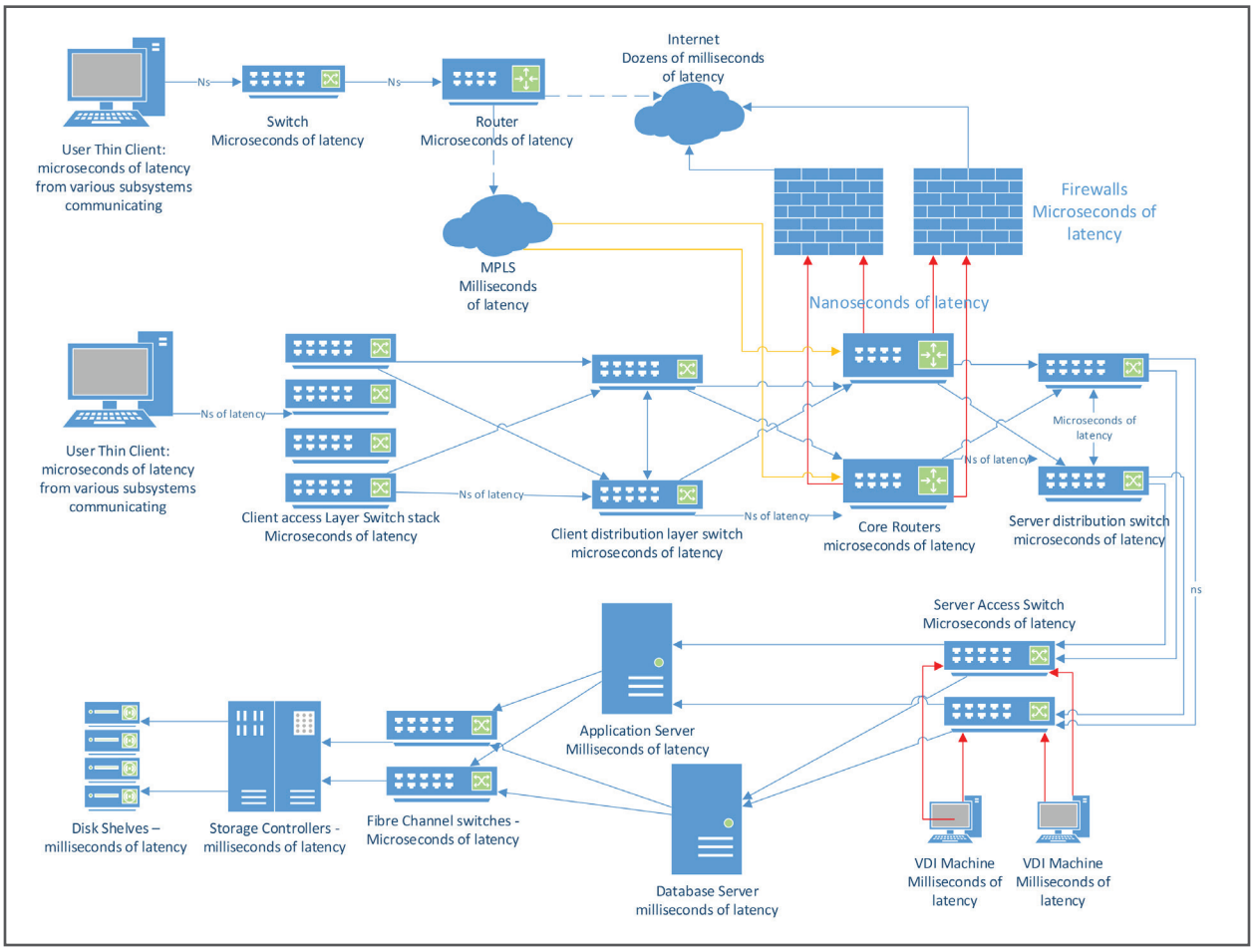


VDI/Remote Desktop/Terminal Servers

Here's one of the places VDI actually has a good business case to help you (there are a LOT of places it doesn't). VDI essentially moves that desktop that your user is interacting with into the datacenter, thereby removing most of the client from server latency interactions. That thick app or direct database connection now has the same speed connection to the servers, whether the user is in California or Calcutta -- just a few microseconds. This can greatly reduce the perceived latency of the connection between the two. As with every VDI discussion, though, there are downsides.

If the latency between your client endpoint and their VDI machine is appreciable, the end user will now not only experience latency between their requesting operations and getting a response; rather, they'll also have latency every time they wiggle the mouse, or type a word. This type of latency quickly causes a type of anger that no helpdesk person can satiate, and justifiably so. When you can't even write a Word document or work in Excel because of a latent mouse or keyboard, it's quite infuriating. This latency becomes a real killer for a lot of VDI projects with highly mobile users who are on cellular networks and otherwise unreliable connectivity. With our latency toleration for VDI already being much lower (you can burst above 100 milliseconds some with other apps and it won't be horrible, but a jittery mouse causes anger), adding unpredictably latent networks to the mix can really cause issues to rise up quickly. Unpredictable access patterns and this end-user experience is one of the reasons that VDI and SSD often go hand-in-hand in discussions.

In our earlier chart of latency localities, there was one thing that clearly stood apart from the rest of the latency definitions: storage. Storage routinely has its latency measured in milliseconds, which is often okay and is not a problem for a lot of environments. With VDI, however, users are pushing more IOPS than most workloads, and the responsiveness of the machine has a direct feedback to that user. If a file server experiences a 200 millisecond latency for a brief period, that's often buffered away from the end user. If a VDI machine experiences that same latency, then the user WILL feel it. Minimizing that worst case latency and helping users' experience is one of the reasons that VDI has pushed so heavily towards SSDs. Taking those milliseconds of latency out of the loop also has another effect: it builds up the buffer for latency elsewhere. Say we're using that 100ms responsiveness window that was mentioned above. A user clicks something, and the response should be within 100ms in order to not be noticed. If our storage is giving us 10-15ms of latency, that's a substantial chunk of our overall 100ms budget. Take that down to one or two milliseconds with faster storage, and we just got eight to 14 percent of our latency budget back to be squandered by the other departments elsewhere. This directly perceivable latency is something that can cause a LOT of problems with VDI, as any additional latency WILL be felt by the user.



The Cloud

Here it is! If there's one thing we know about IT, it's that every single problem we encounter can be fixed with 100 percent efficiency and effectiveness for .01 percent as much money by simply putting it in a cloud. BOOM!

Everything is fixed, our panacea is applied. There's nothing more to write here, let's move on.

Publicly hosted servers and applications:

Assume that you wanted to put your servers or workloads out into a publicly hosted environment that is privately segmented so that only you can access them. There are several potential benefits to this, all revolving around not having to have physical servers and pay for things; however, there are some key things to keep in mind that may cause additional problems, such as (you guessed it) latency. Where may we encounter latency with this public hosting? The virtualization layer that applications are sitting on, the connectivity between different types of applications hosted on it, the storage, the edge connectivity of that provider, the internet between you and it, and our edge connectivity.

The key to the virtualization layer, the storage, and the networking within that publicly hosted server environment, is that you have zero control. There are hardly any SLAs in place with publicly hosted servers, and the SLAs that ARE in place often repay you with more of the service that failed you in the first place! Furthermore, how do you PROVE that the issue you're having with latency is their fault? Most of the tools that control the environment and monitor that kind of thing are internal to the company, not customer facing. But let's assume that part is all taken care of properly, as we've already covered the potential downsides of those problems above. Geography and connectivity are the more infrequently noticed latency issues with publicly hosted apps.

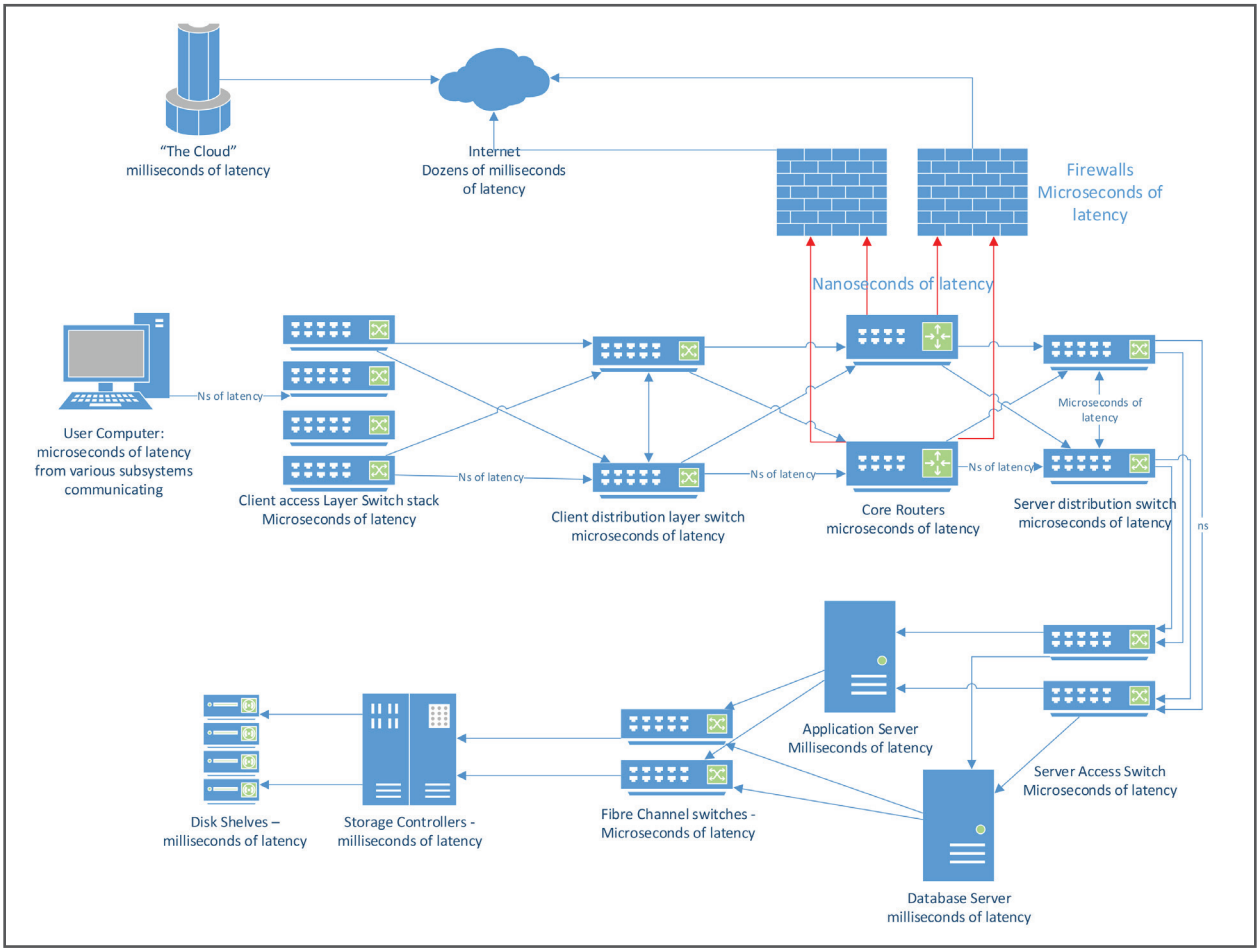
One of the biggest listed benefits of publicly hosted servers and applications is that they can float from datacenter to datacenter without impact to the user. They are also backed up and ready to be spun up at another datacenter in the event that there's a problem (assuming you're paying for this type of service). There is one problem with this, however. What happens when my application server ends up in a different datacenter from my database? I've now added the latency described above with the "chatty apps." What happens when my entire environment floats to another datacenter? Now, I've potentially moved from Nashville to California, adding an additional forty or more milliseconds of latency to every interaction.

"What do you mean everything is slow today? Nothing has changed since yesterday, there's no reason anything should be slower!" Most of the modern hosting providers are offering solutions to these problems with "pin your data to a geography" type solutions, which get it in a rough quadrant of the US. This can be helpful, but still doesn't eliminate the problem as Florida and New York are still a long way away from each other.

Have you considered running a hybrid environment? Make sure you know EXACTLY what talks to what. If you only move your front-end web servers to the publicly hosted environment, leaving something they talk to behind, you just created a huge latency/bandwidth bottleneck in your path. Rather than being locally connected with 10 Gbit networking at a few microseconds of latency, now it's a couple hundred megs at tens of milliseconds. You'll feel that. And speaking of that connectivity...

Say goodbye to one millisecond pings to your servers; they're all now separated from you by the wonderful World Wide Web and a VPN endpoint device. Everyone is now truly equal, as branch offices and main office are all having to traverse additional and potentially latent connections to get to the servers. Let's not forget about those internet pipes too; they have to now be large enough to accommodate ALL client functionality – and not just internet based, but every client-server interaction, AND redundant, AND accommodate for spikes.

Remember that complicated job of the networking people that we were talking about earlier? Your internet and edge will now be the most important thing you have. In case it hasn't been mentioned enough, what happens when a pipe fills up? Latency increases. None of this is to say that publicly hosted servers and applications are bad things; they aren't. If there's one thing we've learned from watching companies like Google and Netflix, it's that they can do WONDERS for your company's growth abilities and flexibility. However, you have to be extremely cognizant of how your pieces and parts interact to make it successful (just from a latency standpoint, let's not even talk about the rest of it).



Conclusion

So in conclusion, what has this very long journey really been for? Simply, to make people stop and consider their environment, their needs, and their expectations. How long is one millisecond? How much is happening in that one millisecond of time that's elapsing? What impact does this one routing change really cause? What kind of performance can I really expect from our new branch office in Australia? Why does my user have a bad experience when they're in Japan with a 100 mbit pipe, but not have issues when they're home with a one mbit pipe? Why did turning on that one feature on our core cripple our perceived performance? Why did our investment in SSDs not fix all of our performance problems? Why didn't "the Cloud" initiative fix all of our problems!?! Step through every operation and consider how long each and every step will take your environment to complete. Odds are, despite all the different possible causes of latency, you will find that there's only one or two things that are causing all of your performance problems.